CHAPTER

12

WINDOWS 2000 APPLICATION SUPPORT

After reading this chapter and completing the exercises, you will be able to:

- Understand the run-time environments and application support in Windows 2000
- Deploy DOS, Win16, OS/2, and POSIX applications
- Fine-tune the application environment for DOS and Win16

In this chapter, you encounter the pieces of the Windows 2000 operating system that endow it with its outstanding power and flexibility. Its numerous run-time environments include support for DOS and 16-bit Windows applications, as well as more modern 32-bit Windows applications, and more limited support for OS/2 (an operating system for PCs developed originally by Microsoft Corporation and IBM, but sold and managed solely by IBM) and POSIX (a portable open systems environment based on UNIX that is mandated in all operating systems that the U.S. government will purchase). You'll have a chance to examine these various subsystems and understand how they work.

Windows 2000 System Architecture

Fundamentally, the Windows 2000 operating system has three main components: the environment subsystem, Executive Services, and user applications (see Figure 12-1).

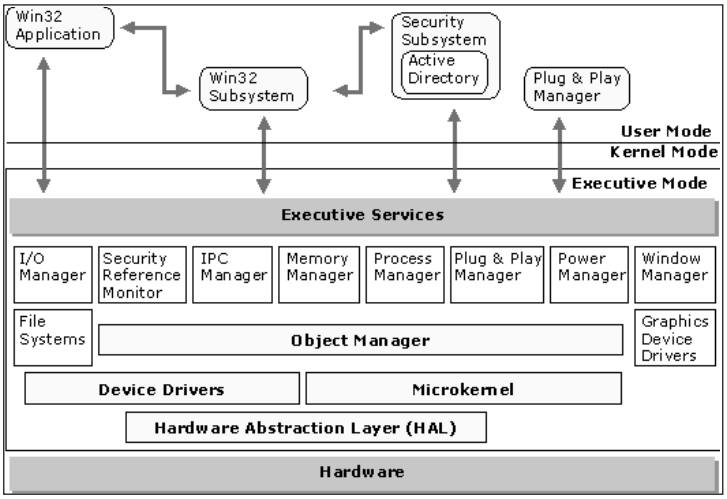

Figure 12-1 Windows 2000 architecture

- **Environment subsystems** offer run-time support for a variety of different kinds of applications under the purview of a single operating system. A **subsystem** is an operating environment that emulates another operating system (such as OS/2 and POSIX) to provide support for applications created for that environment. Just like the applications they support, Windows 2000 environment subsystems run in **user mode**, which means that they must access all system resources through the operating system's **kernel mode**.
- Windows 2000 **Executive Services** and the underlying Windows 2000 **kernel** define the kernel mode of this operating system and its run-time environment. Kernel mode components are permitted to access system objects and resources more or less directly, and provide the many services and access controls that allow multiple users and applications to coexist and interoperate effectively and efficiently.
- User applications provide the functionality and capabilities that make Windows 2000 the most powerful network operating system in use today. All such applications run within the context of an environment subsystem in Windows 2000 user mode. Applications and the subsystems in which they run have a mediated relationship because the client application asks the subsystem to do things for it, and the subsystem complies.

To understand how these components fit together, we need to revisit the concept of processes and threads, building on concepts introduced in Chapter 1.

Kernel Mode Versus User Mode

Before delving further into the architecture of Windows 2000, we'd better make clear the distinction between the Windows 2000 kernel mode and user mode. The main difference between the two modes lies in how memory is used by kernel-mode components and user-mode components.

In user mode, each process perceives the entire 4 GB of virtual memory available to Windows 2000 as its exclusive property—with the condition that the upper 2 GB of addresses are always reserved for system use. This perception remains unaltered, no matter what kind of hardware Windows 2000 may run on. Note also that this address space is entirely virtual, and must operate within the confines of whatever RAM is installed on a machine and the amount of space reserved for the paging file's use. Although the theoretical upper limit for Windows 2000 addresses may be 2 GB (or 4 GB, for system purposes), the real upper limit for Windows 2000 addresses will always be the sum of physical RAM size plus the amount of space in the paging file.

Although processes that operate in user mode may share memory areas with other processes (for fast message passing or sharing information), by default, they don't. This means that one user-mode process cannot crash another, or corrupt its data. This is what creates the appearance that applications run independently, and allows each one to operate as if it had exclusive possession of the operating system and the hardware it controls.

If a user-mode parent process crashes, it will, of course, take its child processes down with it. (Parent and child processes are discussed later in this chapter.)

Processes running in user mode cannot access hardware or communicate with other processes directly. When code runs in the Windows 2000 kernel mode, on the other hand, it may access all hardware and memory in the computer. Thus, when an application needs to perform tasks that involve hardware, it calls a user-mode function that ultimately calls a kernel-mode function.

Because all kernel-mode operations share the same memory space, one kernel-mode function can corrupt another's data and even cause the operating system to crash. This is the reason why the environment subsystems contain as much of the operating system's capabilities as possible, making the kernel itself less vulnerable. For this reason, some experts voiced concern about the change in the Windows 2000 design that moved graphics handlers to the kernel. But because those graphics components were originally part of the Win32 environment subsystem—which must be available for Windows 2000 to operate properly—a crash in either implementation could bring down the system. That's why this change has had little effect on the reliability or stability of Windows 2000.

For a review of the user mode and kernel mode architecture of Windows 2000, please refer to Chapter 1.

Processes and Threads

From a user's point of view, the operating system exists to run programs or applications. But from the view of the Windows 2000 operating system itself, the world is made of processes and threads. A **process** defines the operating environment in which an application or any major operating system component runs. Any Windows 2000 process includes its own private memory space, a set of security descriptors, a priority level for execution, processor affinity data (that is, on a multiprocessor system, information that instructs a process to use a particular CPU), and a list of threads associated with that process. A list of currently active processes can be seen on the Processes tab of the Task Manager (see Figure 12-2). You access the Task Manager by pressing Ctrl+Alt+Delete and clicking the Task Manager button.

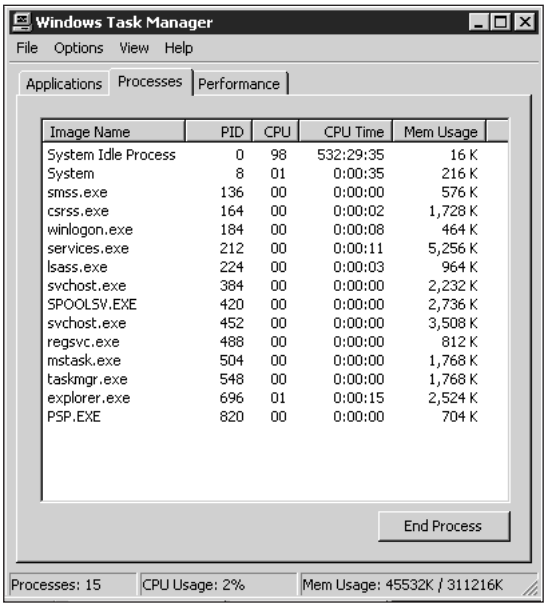

Figure 12-2 The Task Manager Processes tab

The basic executable unit in Windows 2000 is called a **thread**, and every process includes at least one thread. A thread is placeholder information associated with a single use of a program that can handle multiple concurrent users. Within a multithreaded application, each distinct task or any complex operation can be implemented in a separate thread. This explains how Microsoft Word, for instance, can perform spelling and grammar checks in the background while you're entering text in the input window—one thread manages handling input and another performs these checks.

Applications must be explicitly designed to take advantage of threading. Although it's safe to assume that most new 32-bit Windows applications—and the Windows 2000 operating system itself—have been built to use the power and flexibility of threads, older 16-bit Windows and DOS applications are usually single-threaded. Also, it's important to understand that threads are associated with processes and do not exist independently. Processes themselves

don't run, they merely describe a shared environment comprising memory, variables, and other system resources; threads represent those parts of any program that actually run.

Processes can create other processes, called **child processes**, and those child processes can inherit some of the characteristics and parameters of the **parent process**. (A child process is a replica of the parent process and shares some of its resources, but cannot exist if the parent is terminated.) This usually works as follows:

- When a user logs on to Windows 2000 successfully, a shell process is created inside the Win32 subsystem within which the logon session operates. The Win32 subsystem is an operating environment that supports 32-bit Windows applications and is required to run Windows 2000. This process is endowed with a security token used to determine if subsequent requests for system objects and resources may be permitted to proceed. This shell process defines the Win32 subsystem as the parent process for that user.
- Each time a user launches an application or starts a system utility, a child process is created within the environment subsystem where that application or utility must run. This child process inherits its security token and associated information from the user account, but is a child of the environment subsystem within which it runs. This "dual parentage" (security information from the user account and, run-time environment from the environment subsystem) explains how Windows 2000 can run multiple kinds of applications, yet maintain consistent control over which system objects and resources any user process is permitted to access.

For example, each of the environment subsystems discussed in the following sections is an executable file—a combination of processes and threads running within the context of those processes (a **context** is the current collection of Registry values and run-time environment variables in which a process or thread runs). When an application runs in a Windows 2000 subsystem, it actually represents a child of the parent process for the environment subsystem, but one that is endowed with the permissions associated with the security token of the account that launches the process. Whenever a parent process halts or is stopped, all child processes stop as well.

Environment Subsystems

Windows 2000 offers support for various application platforms. Although primarily designed for 32-bit Windows applications, Windows 2000 includes support for backward compatibility for 16-bit Windows and DOS applications. To comply with U.S. government purchasing standards, Windows 2000 also includes basic support for OS/2 and POSIX applications.

Windows 2000 support for multiple run-time environments, also known as environment subsystems, confers numerous advantages, including:

- It permits users to run more than one type of application concurrently, including 32-bit Windows, 16-bit Windows, and DOS applications, as well as OS/2 and POSIX applications.

- It makes maintaining the operating system easier, because the modularity of this design means that changes to environment subsystems require no changes to the kernel, as long as interfaces remain unchanged.
- Modularity makes it easy to add or enhance Windows 2000—if a new OS is developed in the future, Microsoft could decide to add a subsystem for that OS to Windows 2000 without affecting other environment subsystems.

The catch to using an architecture that supports multiple environment subsystems is in providing mechanisms to permit those subsystems to communicate with one another when necessary. In the Windows 2000 environment, each subsystem runs as a separate user-mode process, so that subsystems cannot interfere with or crash one another. The only exception to this insulation effect occurs in the Win32 subsystem: because all user-mode I/O passes through this subsystem, the Win32 subsystem must be running for Windows 2000 to function. If the Win32 subsystem's process ends, the whole operating system goes down with it. That is *not* true for the POSIX or OS/2 subsystems, which may be shut down without affecting anything but their child processes.

Applications and the subsystems in which they run have a client/server relationship, in that the client application asks the server subsystem to do things for it, and the subsystem complies. For example, if a Win32 client application needs to open a new window (perhaps to create a Save As dialog box), it doesn't create the window itself, but asks the Win32 subsystem to draw the window on its behalf.

The client issues the request via a mechanism known as a **local procedure call (LPC)**. The serving subsystem makes its capabilities available to client applications by linking them to a **dynamic link library (DLL)**. You could think of a DLL as a set of buzzers, where each one is labeled with the capabilities it provides. Pushing a specific buzzer tells the server subsystem to do whatever the label tells it to. This form of messaging is transparent to the client application (as far as it knows, it's simply calling a procedure). When a client pushes one of those buzzers (requests a service), it appears as if the act is handled by the DLL; no explicit communication with a server subsystem is needed. If a service isn't listed in the library, an application can't request it; thus, a word processor running in a command-line environment inside the OS/2 subsystem, for example, can't ask that subsystem to draw a window.

The inability to provide functions that aren't offered by a subsystem can occasionally present problems. For example, the POSIX subsystem includes no networking capabilities, so POSIX applications can't access the network. Microsoft has no plans to change this arrangement, because doing so would complicate the subsystem design.

Message passing is a fairly time-consuming operation, because any time the focus changes from one process to another, all the information for the calling process must be unloaded and replaced with the information for the called process. In operating system lingo, this change of operation focus from one process to another is called a **context switch**. To permit the operating system to run more efficiently, Windows 2000 avoids making context

switches whenever possible. To that end, Windows 2000 includes the following efficiency measures:

- It caches attributes in DLLs to provide an interface to subsystem capabilities, so that (for example) the second time Microsoft Word requests a window to be created, this activity may be completed without switching context to the Win32 subsystem.
- It calls Executive Services (the collection of kernel mode Windows 2000 operating system components that provides basic system services such as I/O, security, object management, and so forth) directly, to perform tasks without requesting help from an underlying environment subsystem. Because the kernel is always active in another process space in Windows 2000, calling for kernel-mode services does *not* require a context switch.
- It batches messages so that when a server process is called, several messages can be passed at once—the number of messages has no impact on performance, but a context switch does. By batching messages, Windows 2000 allows a single context switch to handle multiple messages in sequence, rather than requiring a context switch for each message.

When LPCs must be used, they're handled as efficiently as possible. Likewise, their code is optimized for speed, and special message-passing functions can be used for different situations, depending (for example) on the size of the messages passed, or the circumstances in which they're sent.

So far, we've covered the broad view of how environment subsystems interact with client applications. Now, let's take a closer look at these subsystems.

The Win32 Subsystem

As the only subsystem required for the functioning of the operating system, the **Win32 subsystem** handles all major interface capabilities. In early versions of Windows NT, the Win32 subsystem included graphics, windowing, and messaging support, but since Windows NT 4.0, and in Windows 2000, these have been moved to the kernel and are now part of Executive Services.

In Windows 2000, user mode components of the Win32 subsystem consist of the console (text window support), shutdown, hard-error handling, and some environmental functions to handle such tasks as process creation and deletion. The Win32 subsystem is also the foundation upon which **virtual DOS machines (VDMs)** rest; these permit Windows 2000 to deliver both DOS and Win16 subsystems, so that DOS and Win16 applications can run on Windows 2000 unchanged (we'll talk more about VDMs and the DOS and Win16 subsystems later in this chapter). Try Hands-on Project 13-1 to launch a Win16 application in its own address space.

The OS/2 Subsystem

Unlike the Win32 subsystem, which starts on system startup, the **OS/2 subsystem** only begins when a user launches an application that requires its services, at which point it remains resident in memory until the system is shut down and restarted—logging off and logging back on again will not restart the OS/2 subsystem.

The Windows 2000 OS/2 subsystem is limited in several ways. The OS/2 subsystem is limited to OS/2 version 1.x, so, out of the box, this subsystem can run only command-line OS/2 applications.

The POSIX Subsystem

POSIX (Portable Operating System Interface for UNIX) is a set of standards owned by the IEEE (Institute of Electrical and Electronics Engineers) that defines various aspects of an operating system. So far, only one of those standards has been adopted: POSIX.1. The Windows 2000 **POSIX subsystem** (Posix.exe) is POSIX.1-compatible.

POSIX defines only API (application programming interface) calls between applications and the operating system, so in general, any application written for POSIX must rely on other operating systems for functions such as security and networking. Also, any POSIX application that accesses files must have access to an NTFS partition, because NTFS provides functionality that FAT cannot deliver and that POSIX applications need (such as the ability to support multiple names for a single data file). Recently, POSIX.1 and POSIX.2 interfaces were included into a somewhat larger interface known as the X/Open Programming Guide 4.2.

WIN32 APPLICATIONS

So far, we've examined the components of the Windows 2000 operating system kernel. Now, it's time to see how applications run under that operating system.

The Environment Subsystem

As we've mentioned, the Win32 subsystem is the main environment subsystem under Windows 2000, and the only one required for operation. Strictly speaking, even the other environment subsystems (such as OS/2 and POSIX) are Win32 emulation applications that run as child processes to the main Win32 process, although they are full-blown operating systems and not just application environments like the virtual DOS machines (VDMs) that run under Win32 to support DOS applications. (We will explain VDMs and DOS application support in more detail later in this chapter.)

Multithreading

When a program's process contains more than one thread of execution, it's said to be a **multithreaded process**. The main advantage of multithreading is that it provides multiple threads of execution within a single memory space without requiring that messages be passed between processes or that local procedure calls be used, thus simplifying thread communication. Threads are easier to create than processes because they don't require as much context information, nor do they incur the same kind of overhead when switching from one thread to another within a single process.

Some multithreaded applications can even run multiple threads concurrently among multiple processors (assuming a machine has more than one). One more advantage to threading is that it's *much* less complicated to switch operation from thread to thread than to switch from

one process to another. That's because every time a new process is scheduled for execution, the system must be updated with all the process's context information. Also, it's often necessary to remove one process to make room for another, which may require writing large amounts of data from RAM to disk for the outgoing process, before copying large amounts of data from disk into RAM to bring in the incoming process.

As a point of comparison, a thread switch can normally be completed in somewhere between 15 and 25 machine instructions, whereas a process switch can take many thousands of instructions to complete. Because most CPUs are set up to handle one instruction for every clock cycle, this means that switching among threads is hundreds to thousands of times faster than switching among processes.

The big trick with multithreading, of course, is that the chances that one thread could overwrite another are increased with each additional thread, so this introduces the problem of protecting shared areas of memory from intraprocess thread overwrites. Windows 2000 handles this by managing access to memory carefully, and limiting which sections of memory any individual thread can write to by locking them, as you'll see in the next section.

Memory Space

Multithreaded programs must be designed so that threads don't get in each other's way, and they do this by using Windows 2000 **synchronization objects**. A section of code that modifies data structures used by several threads is called a **critical section**. It's very important that a critical section never be overwritten by more than one thread at once. Thus, applications use Windows 2000 synchronization objects to prevent this from happening, creating such objects for each critical section in each process context. When a thread needs access to a critical section, the following occurs:

1. A thread requests a synchronization object. If it is unlocked (not suspended in a thread queue), the request proceeds. Otherwise, go to Step 2.
2. The thread is suspended in a thread queue until the synchronization object is unlocked for its use. As soon as this happens, Windows 2000 releases the thread and locks up the object.
3. The thread accesses the critical section.
4. When the thread is done, it unlocks the synchronization object so that another thread may access the critical object.

Thus, multithreaded applications avoid accessing a single data structure with more than one thread at a time by locking its critical section when it's in use and unlocking it when it's not.

Input Message Queues

One of the roles of the Win32 subsystem is to organize user input and get it to the thread to which that input belongs. It does this by taking user messages from a general input queue, and distributing them to the **input message queues** for the individual processes.

As we'll discuss later in this chapter, Win16 applications normally run within a single process, so they share a message input queue, unlike Win32 or DOS applications with their individual queues.

Base Priorities

When a program is started under Windows 2000, its process is assigned a particular priority class, generally Normal—but there is a range of options (see Figure 12-3). The priority class helps determine the priority at which threads in a process must run, on a scale from 0 (lowest) to 31 (highest). In a process with more than one active thread, each thread may have its own priority, which may be higher or lower than that of the original thread, but that priority is always relative to the priority assigned to the underlying process, which is known as the **base priority**. Managing priorities may be accomplished in one of several ways, and can sometimes provide a useful way to improve application performance. These include Task Manager and the Start command, as discussed in Chapter 11.

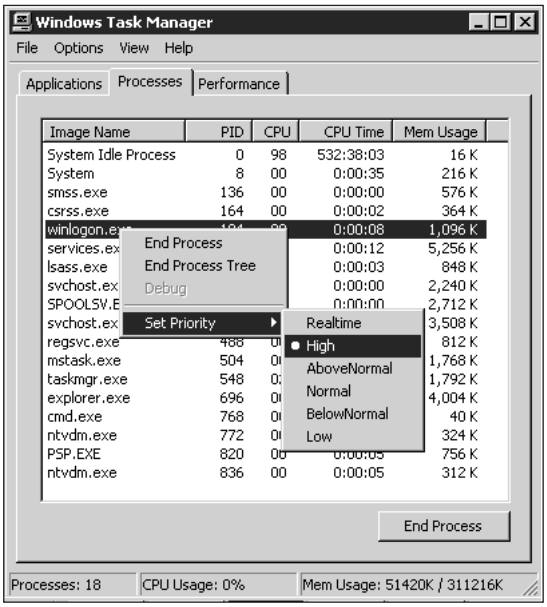

Figure 12-3 The Task Manager Processes tab showing priority options

DOS and the Virtual DOS Machine

DOS and Win16 applications work somewhat differently from Win32 applications. Rather than each running in the context of its own process, these applications run a virtual DOS machine (VDM), a special environment process that simulates a DOS environment so that non-Win32 Windows applications can run under Windows 2000. In fact, it's reasonable to describe two separate operating environments that can run within a VDM: one supports

straightforward DOS emulation and may be called the **DOS operating environment**; the other supports operation of Win16 applications within a VDM, and may be called the **Win16 operating environment**.

The DOS operating environment under Windows 2000 is established by a Win32 process named Ntvdm.exe (see Figure 12-4). In fact, if you look at the Processes tab of the Task Manager when a DOS application is active, you'll see this process. Ntvdm creates the environment where DOS applications execute. Each DOS application that is launched is executed within a separate emulation environment. Thus, if you launch three DOS applications, three instances of Ntvdm will appear in the process list. Once a DOS application terminates, Windows 2000 also shuts down the emulation environment for that application by terminating the associated instance of Ntvdm. This frees the system resources to be used elsewhere and is unlike the behavior of VDMs in Windows NT 4.0.

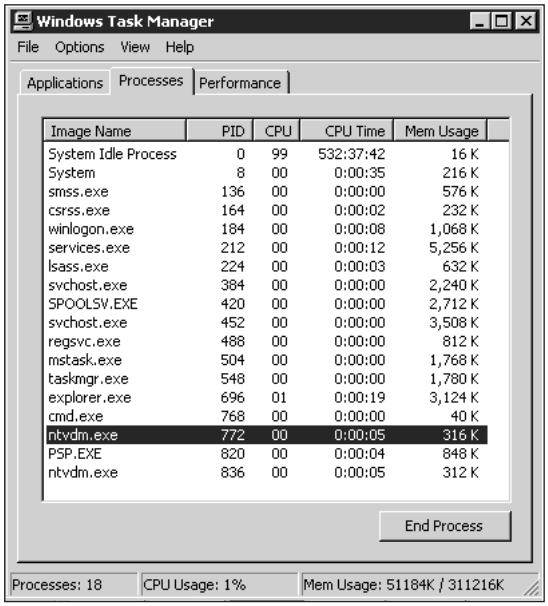

Figure 12-4 The Task Manager Processes tab showing Ntvdm.exe

The environment created in a VDM is not the same as that available to Win32 applications. Instead, it's equivalent to the environment of Windows 3.x Enhanced mode, in which each DOS application has access to 1 MB of virtual memory, with 1 MB of extended memory and expanded memory if necessary.

By default, all DOS applications run in their own VDM. By default, all Win16 applications share a single VDM (just as they do in "real Windows 3.x" environments).

VDM Components

The VDM runs using the following files:

- *Ntio.sys:* The equivalent of Io.sys on MS-DOS machines, runs in **real mode** (real mode is a mode of operation for x86 CPUs wherein they can address only 1 MB of memory, broken into 16 64-KB segments). It provides "virtual IO" services to the DOS or Win16 applications that run in a VDM.
- *Ntdos.sys:* The equivalent of Msdos.sys, runs in real mode. It provides basic DOS operating system services to the DOS or Win16 applications that run in a VDM.
- *Ntvdm.exe:* A Win32 application that runs in kernel mode. This is the .exe file that provides the run-time environment within which a VDM runs. If you look at the list on the Processes tab of Task Manager, you'll see one such entry for each separate VDM that's running on your machine.
- *Ntvdm.dll:* A Win32 dynamic link library that runs in kernel mode. Ntvdm.dll provides the set of procedure stubs that fool DOS and Win16 programs into thinking they're talking to a real DOS machine with exclusive access to a PC, when in fact they're communicating through a VDM with Windows 2000.
- *Redir.exe:* The virtual device driver (VDD) redirector for the VDM. This software forwards I/O requests from programs within a VDM for I/O services through the Win32 environment subsystem to the Windows 2000 I/O Manager in Executive Services. Whenever a DOS or Win16 program in a VDM thinks it's communicating with hardware, it's really communicating with Redir.exe.

Virtual Device Drivers

DOS applications do not communicate directly with Windows 2000 drivers. Instead, a layer of **virtual device drivers (VDDs)** underlies these applications, and they communicate with Windows 2000 32-bit drivers. Windows 2000 supplies VDDs for mice, keyboards, printers, and communication ports, as well as file system drivers (which include one or more network drivers, each of which is actually implemented as a file system driver).

AUTOEXEC.BAT and CONFIG.SYS

When a DOS application is started, Windows 2000 runs the files specified in the application's program information file (PIF) or in AUTOEXEC.NT (see Figure 12-5) and CONFIG.NT (see Figure 12-6), the two files that replace AUTOEXEC.BAT and CONFIG.SYS for VDMs. AUTOEXEC.NT installs CD-ROM extensions and the network redirector, and can even provide DOS Protected Mode Interface (DPMI) support, to permit DOS and Win16 applications to access more than 1 MB of memory within a virtual (or real) DOS machine. CONFIG.NT loads into an upper memory area for its VDM, and can support HIMEM.SYS to enable extended memory; it also sets the number of files and buffers available to DOS or Win16 programs, and provides necessary details to configure expanded memory. Try Hands-on Project 12-4 to explore AUTOEXEC.NT and CONFIG.NT.

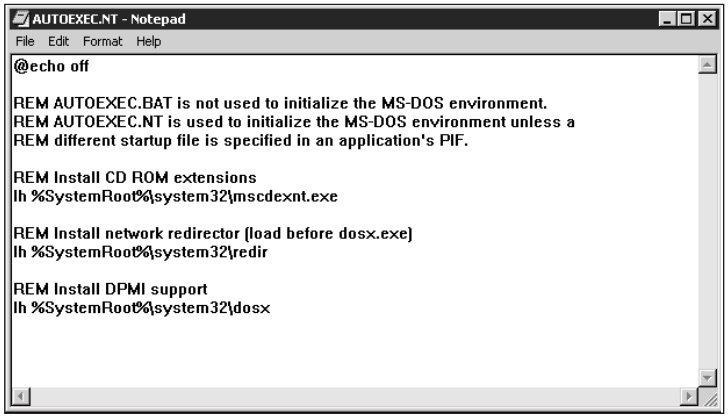

Figure 12-5 AUTOEXEC.NT viewed through Notepad

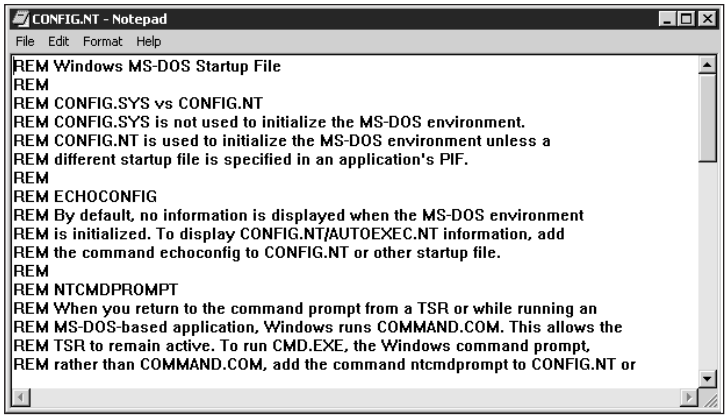

Figure 12-6 CONFIG.NT viewed through Notepad

CONFIG.SYS isn't used at all by Windows 2000, whereas AUTOEXEC.BAT is only used at system startup to set path and environment variables for the Windows 2000 environment. Neither file is consulted when it comes to running applications or initializing drivers; those settings must exist in the system Registry to work at all.

Once read from AUTOEXEC.BAT, path and environment variables are copied to the Registry, to HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment (see Figure 12-7).

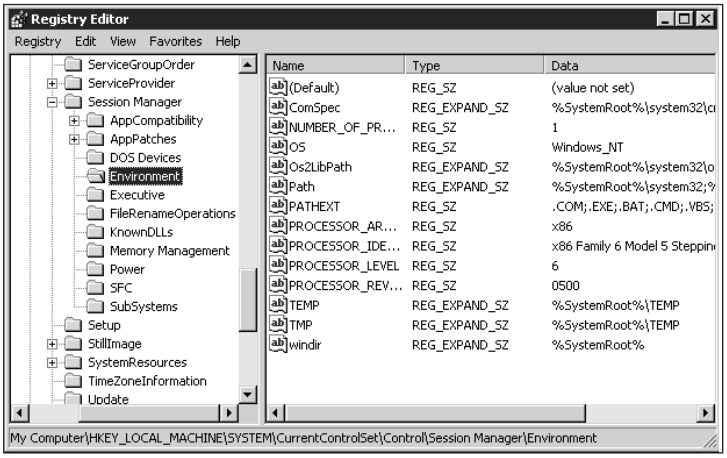

Figure 12-7 Regedit displaying the Environment subkey

Custom DOS Environments

Windows 2000 offers customizable environmental controls for the DOS environment. These controls can be used to fine-tune or simply alter how a DOS application functions. To customize a DOS application's execution parameters, open the Properties dialog box for that executable (.exe or .com) file. This is performed by right-clicking over an executable file and selecting Properties from the resulting menu. Try Hands-on Project 12-2 to explore the properties of a DOS application.

The Properties dialog box for a DOS (FAT 16) executable file has six tabs. The General tab lists the same data items as any other file within the Windows 2000 environment (see Figure 12-8). The Program tab (see Figure 12-9) offers controls over:

- *Filename*—The name of the file
- *Command line*—Used to add command-line parameter syntax
- *Working*—Used to define the working directory, which is the directory from which the application will load files and where it saves files
- *Batch file*—Used to run a batch file before launching the executable file
- *Shortcut key*—Used to define a keystroke that will launch the executable file
- *Run*—Used to define the window size of the DOS environment—normal, maximized, or minimized
- *Close on exit*—Informs the OS to close the DOS window when the application terminates
- *Advanced* button—Allows you to define the path to alternate AUTOEXEC.NT and CONFIG.NT files
- *Change icon* button—Changes the icon displayed for the executable file

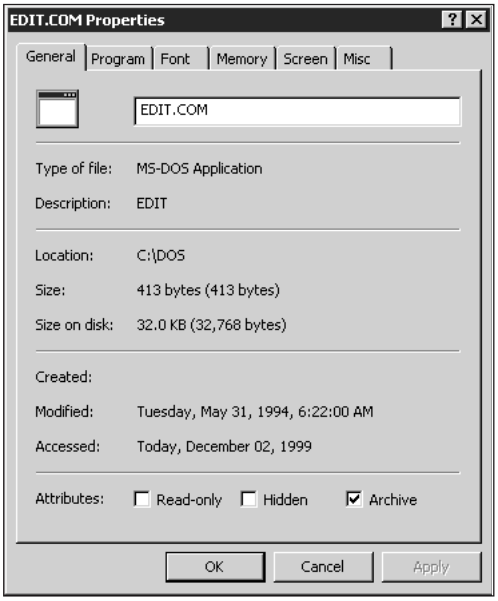

Figure 12-8 EDIT.COM Properties, General tab

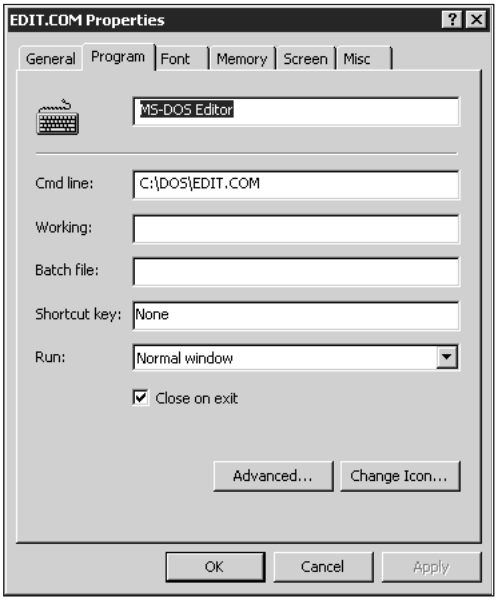

Figure 12-9 EDIT.COM Properties, Program tab

The Font tab is used to define the font used by the DOS application (see Figure 12-10). The Memory tab is used to define the memory parameters of the DOS environment (see Figure 12-11). These controls include settings for conventional memory, expanded memory (EMS),

extended memory (XMS), and protected-mode (DBMI) memory. The Screen tab is used to define whether the DOS application loads full-screen or in a window, to emulate fast ROM, and to define whether to allocate dynamic memory (see Figure 12-12). The Misc tab is used to allow a screen saver over the DOS window and to define whether the mouse is used by the DOS application, if the DOS application is suspended when in the background, whether to warn if the DOS application is active when you attempt to close the DOS window, how long the application will wait for I/O before releasing CPU control, whether to use fast pasting (a quick method for pasting information into the application; this doesn't work with some programs, so disable this check box if information does not paste properly), and which Windows shortcut keys will be reserved for use by Windows 2000 instead of the DOS application (see Figure 12-13).

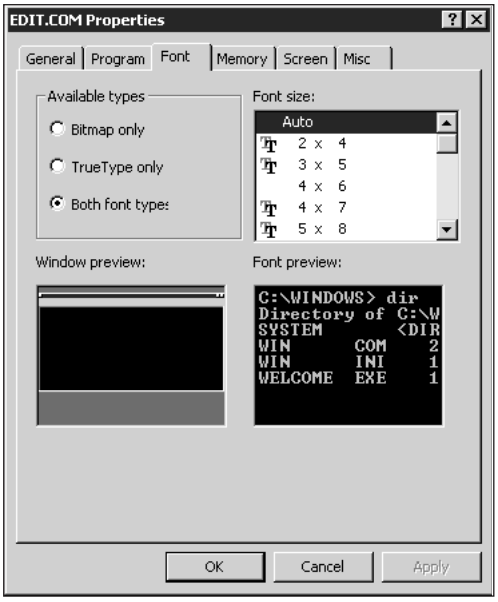

Figure 12-10 EDIT.COM Properties, Font tab

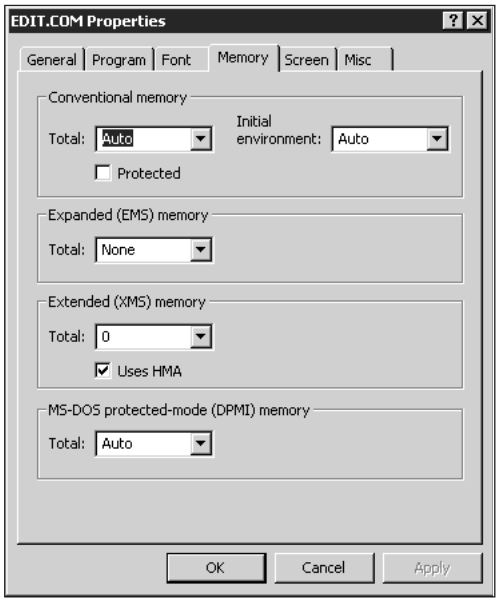

Figure 12-11 EDIT.COM Properties, Memory tab

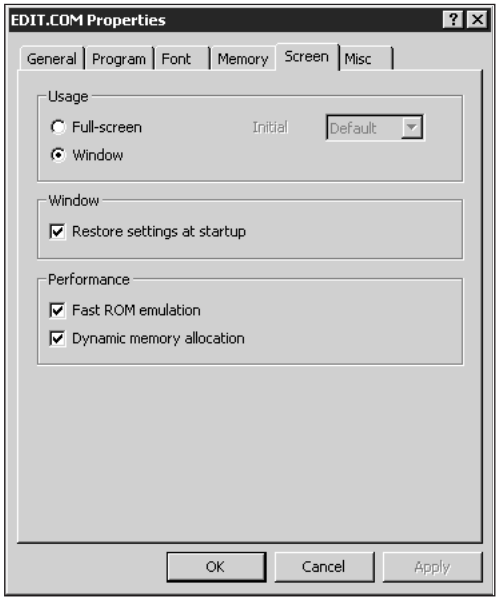

Figure 12-12 EDIT.COM Properties, Screen tab

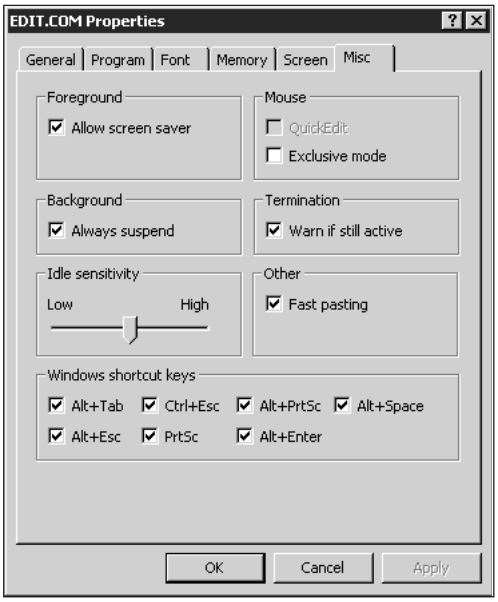

Figure 12-13 EDIT.COM Properties, Misc tab

Once you alter any portion of one of these tabs, a new shortcut for the application is created that will retain these changes. Thus you can reuse and fine-tune your custom DOS environment settings for each application.

Win16 Concepts and Applications

Like DOS applications, Win16 applications also run in a VDM, although unlike DOS applications, which by default run in their own individual address spaces, all Win16 applications run in the same VDM unless specified otherwise. This permits them to act like Win32 applications, and lets multiple Win16 applications interact with one another in a single VDM. This creates the appearance that multiple applications are active simultaneously. (Usually, only one Win16 application in a VDM can be active at any given moment, but this form of **multitasking**—which Microsoft calls cooperative multitasking—creates a convincing imitation of the more robust and real multitasking available to Win32 applications.) The **Win16-on-Win32** VDM—usually called **WOW**—runs as a multithreaded application, with each Win16 application being one thread (see Figure 12-14).

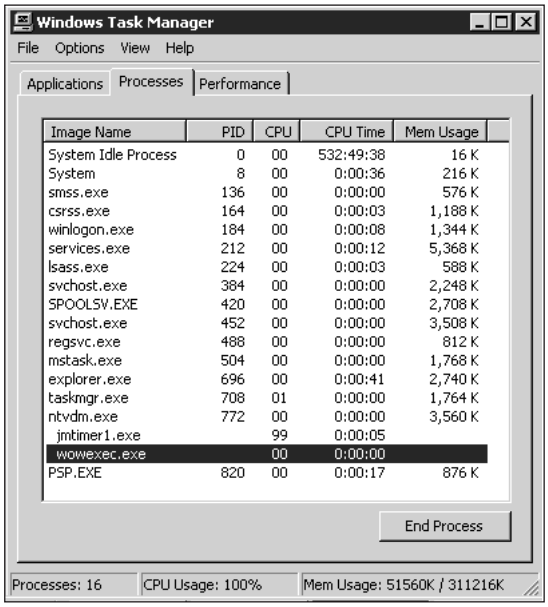

Figure 12-14 The Task Manager Processes tab showing Wowexec.exe

Win16-on-Win32 (WOW) Components

The WOW subsystem has the following components:

- *Wowexec.exe:* Handles the loading of 16-bit Windows-based applications
- *Wow32.dll:* The dynamic link library for the WOW application environment
- *ntvdm.exe*, *ntvdm.dll*, *ntio.sys*, and *redir.exe:* Run the VDM
- *Vdmredir.dll:* The redirector for the WOW environment
- *Krnl386.exe:* Used by WOW on x86-based systems
- *Gdi.exe:* A modified version of Windows 3.10 Gdi.exe
- *User.exe:* A modified version of Windows 3.10 User.exe

Calls made to 16-bit drivers are transferred ("thunked") to the appropriate 32-bit driver, without the application being aware of it. Similarly, if a driver needs to return information to an application, it must be thunked back again. This back and forth translation helps explain why many Win16 applications run more slowly in a VDM on Windows 2000 than they do on other versions of Windows (even Windows 95), where no such translations are required.

Once a WOW environment is created, Windows 2000 will sustain that environment until the system is rebooted or you manually terminate the Wowexec.exe task (such as via the Task Manager Processes tab). Creating new WOW environments each time a Win16 application is launched was deemed more costly in terms of resources and CPU time than maintaining a WOW environment once it has been created throughout a boot session. Thus, if you use

Win16 applications often, this function will offer you some benefit. But if you seldom use Win16 applications, you'd be better off terminating the WOW environment after you finish with the hosted application.

Memory Space

By default, all Win16 applications run as threads in a single VDM process (try Hands-on Project 12-5 to explore the number of threads used by a process). However, it might be a good idea not to permit this, because multiple threads running in a single process can affect the performance of each application. This mixture of applications can also make tracking applications more difficult, because most monitoring in Windows 2000 takes place on a per-process basis, not on a per-thread basis. Finally, running all Win16 applications in a single VDM means that if one of those applications goes astray and causes the VDM to freeze or crash, all applications in that VDM will be affected ("just like real Windows 3.x!").

Separate and Shared Memory

The "lose one, lose them all" effect of a single shared VDM explains why you might choose to run Win16 applications in separate VDMs. That way, you'll increase the reliability of those applications as a whole, and one errant application won't take down all the other Win16 applications if it crashes. Likewise, you'll make preemptive multitasking possible (that is, one busy application won't be able to hog the processor), and you'll be able to take advantage of multiple processors if you have them, because all the threads in a single VDM process must execute on the same processor.

The disadvantages of running Win16 applications in separate memory spaces focus on memory usage and interprocess communications: Each additional process running on a machine requires about 2 MB of space in the paging file, and 1 MB of additional working set size, or the amount of data that the application has in memory at any given time. Also, those older Win16 applications that don't support Dynamic Data Exchange (DDE) or object linking and embedding (OLE) won't be able to communicate with each other. Additionally, running Win16 applications as processes instead of threads increases the time it takes to switch from one application to another, because each such switch requires a full context change from one process to another. The best way to observe the impact of this separation is to try it the default way (whereby all Win16 applications share a single VDM), and then set up those Win16 applications in separate VDMs and compare the performance that results from each such scenario.

To launch a Win16 application in a separate memory space, you must first create a shortcut to the executable. Then edit the properties of the shortcut and select the *Run in separate memory space* check box. You can also start 16-bit applications in their own address space via the /separate command line switch. The proper syntax is: start/separate [16-bit program executable name].

The Run command in Windows 2000 does not have a *Run in separate memory space* check box like Windows NT 4.0; thus, a Win16 application cannot be launched in a separate memory space from the Run command.

Message Queues

As mentioned earlier, the Win32 subsystem is responsible for collecting user input and getting it to those applications that need it. However, unlike Win32 applications, all Win16 applications running in a single process share a message queue. Therefore, if one application becomes unable to accept input, it will block all other Win16 applications in that VDM from accepting further input as well.

Threads

As we mentioned earlier, Win16 threads that run in a VDM do not multitask like threads running in the Win32 subsystem. Instead of being preemptively multitasked, so that one thread can push another aside if its priority is higher, or so that any thread that's been taking up too much CPU time can be preempted, all application threads within a WOW VDM are cooperatively multitasked. This means that any one thread—which corresponds to any Win16 application—can hog the CPU. This is sometimes called the "good guy scheduling" algorithm, because it assumes that all applications will be well behaved and relinquish the CPU whenever they must block for I/O or other system services. The net effect, however, is that WOW VDMs behave as if they have only one execution thread to share among all applications within the VDM.

Using Only Well-Behaved DOS and Win16 Applications

Many DOS applications, as well as numerous older Win16 applications, often take advantage of a prerogative of DOS developers—namely, the ability to access system hardware directly, bypassing any access APIs or drivers that the system might ordinarily put between an application and the underlying hardware. Although such applications will work fine in DOS, Windows 3.x, and even Windows 95, this is not the case with Windows 2000. The division into user mode and kernel mode in Windows 2000 means that any application that attempts to access hardware directly will be shut down with an error message to the effect of "illegal operation attempted."

In Windows 2000 terminology, any application that attempts direct access to hardware is called "ill behaved." Such applications will not run in a VDM. On the other hand, any Win16 or DOS application that uses standard DOS or Windows 3.x APIs instead of attempting direct access to hardware will work in a VDM. Such applications are called well behaved. Unfortunately, there is no list of well-behaved applications available, so the only way to tell the difference is to test the ones you'd like to use with Windows 2000 and see what happens. If an application doesn't perform properly, it shouldn't be deployed on your system. We suggest that you deploy only well-behaved applications for use with Windows 2000, and that you seriously consider replacing any ill-behaved applications you may find in your current collection of programs. Try Hands-on Project 12-3 to explore the effects of VDMs in Windows 2000.

THE OS/2 SUBSYSTEM

The Windows 2000 OS/2 subsystem provides a way to run character-mode OS/2 applications on x86 computers. Unlike the VDMs and WOW, the OS/2 subsystem is indeed a separate operating system, albeit one that runs like a Win32 application. It works just as the OS/2 operating system would if it were running independently, although some of its features (such as its ability to share information between processes) are implemented through Windows 2000, not as they would be in OS/2. One thing that *doesn't* work the same as in the original version of OS/2 is the amount of memory available, because the OS/2 subsystem utilizes the 4 GB of memory normally available to Win32 processes, rather than the 16 MB with which OS/2 was designed to work. Because this represents a substantial improvement over the original, this difference has not occasioned too many complaints.

OS/2 Version 1.x

OS/2 1.x doesn't look much like any version of OS/2 you're likely to see, because it's completely character-based and doesn't include the familiar graphical workplace shell. Character-based applications that call the OS/2 Presentation Manager will not run under Windows 2000. This characteristic makes the OS/2 subsystem of limited use in most organizations, because most OS/2 applications require graphical interface support.

OS/2 Components

The OS/2 subsystem has the following components:

- *Os2ss.exe:* The main component of the OS/2 subsystem and the one started when the first OS/2 application is loaded
- *Os2dll.dll:* Shares address space with each OS/2 application and handles the communication between the application and the subsystem, Os2ss.exe
- *Os2.exe:* A Win32 program that passes the name of the OS/2 application and any command-line parameters to Os2srv.exe. This portion of the OS/2 subsystem only starts once, no matter how many OS/2 applications are running.
- *Os2srv.exe:* The component that actually starts each OS2 application for the OS2 subsystem

These components don't start until the first OS/2 application is started, at which point the OS/2 subsystem comes up, and it isn't shut down until the system is rebooted. Logging off and logging back on to a Windows 2000 machine will not reset the OS/2 subsystem; that requires an actual system shutdown or restart.

Bound Applications

Bound applications, or those that can run under either OS/2 or in a VDM, will run in the OS/2 subsystem on an x86 machine if one is available, because they'll run faster. The only way to prevent an application that can go either way from running within the OS/2 subsystem,

should you wish to do so (perhaps because that application won't run in OS/2), is to run Forcedos.exe. This utility appears in the \System32 directory, and forces bound applications to run in a VDM instead of in the OS/2 subsystem environment.

OS/2 Configuration

When the OS/2 subsystem starts up for the first time, it checks the Registry for OS/2 setup information. If it doesn't find any, it checks the OS/2 CONFIG.SYS. If CONFIG.SYS does not exist, then the subsystem adds shell initialization information to the Registry.

To update the OS/2 configuration information, you must edit CONFIG.SYS with an OS/2 text editor (Notepad won't do; it must be an OS/2 editor). Open the file, which will be a temporary copy of the configuration information in the Registry. Make the changes and save them, and then those changes will be stored in the Registry and will take effect after you next restart your computer.

When configuring OS/2, you can use the commands shown in Table 12-1. If you use a command not found in this table, it will be ignored.

Table 12-1 OS/2 Subsystem Configuration Commands

Command	Description
protshell	Specifies the command interpreter, although only the Windows 2000 interpreter Cmd.exe is supported
devicename	Specifies a user-defined Windows 2000 device driver used by OS/2 applications
libpath	Specifies the location of the 16-bit OS/2 DLLs
set	Sets environment variables
country	Lets you provide a country code that specifies time, date, and currency conventions
codepage	Specifies the code page the system will use
devinfo=KBD	Specifies the information the keyboard needs to use a particular code page

POSIX Subsystem

As already mentioned, POSIX is a set of APIs intended to permit UNIX applications to run on a variety of operating systems without requiring reimplementation. At this point, only one POSIX standard of the 12 proposed has been formalized: POSIX 1.x. Because POSIX is only an API, the list of what POSIX can't do is as long as the list of what it can do. Because POSIX was designed for use in a very simple operating environment, it supports only local input and output, and does not even support networking (although you can get to network-accessible files via other parts of Windows 2000). Although POSIX technically doesn't support printing, you can pipe information to another Windows 2000 subsystem for output if you connect or redirect a serial or parallel port and access it with the *net use* command from the command prompt.

For expanded POSIX support on Windows 2000, you need to deploy a third-party emulation service. Softway Systems, which has been recently acquired by Microsoft, offers Interix, an expanded POSIX environment emulator and suite of development tools. For more information, see *http://www.interix.com/home.html*.

Chapter Summary

- Windows 2000 is divided into three main parts: environment subsystems, Executive Services, and user applications. The environment subsystems provide support for applications written for a variety of operating systems, not just for Windows 2000; the Executive Services define the Windows 2000 run-time environment; and user applications provide additional functionality for a variety of services, such as word-processing and e-mail applications.
- Three environment subsystems (Win32, OS/2, and POSIX) run under Windows 2000, plus two special-purpose operating environments (VDM and WOW) that provide backward compatibility within the Win32 subsystem for Win16 and DOS applications.
- Of these subsystems, only Win32 is crucial to the functioning of Windows 2000 as a whole. The other subsystems only start up as they're needed, but once launched, these subsystems tend to persist until the machine is shut down and restarted.

Key Terms

base priority — The lowest priority that a thread may be assigned, based on the priority assigned to its process.

bound application — An application capable of running under the OS/2 subsystem or in a virtual DOS machine. If the OS/2 subsystem is available, it will be used by default.

child process — A process spawned within the context of some Windows 2000 environment subsystems (Win32, OS/2, or POSIX) that inherits operating characteristics from its parent subsystem, and access characteristics from the permissions associated with the account that requested it to be launched.

context — The collection of Registry values and run-time environment variables in which a process or thread is currently running.

context switch — The act of unloading the context information for one process and replacing it with the information for another, when the new process comes to the foreground.

critical section — In operating system terminology, this refers to a section of code that can only be accessed by a single thread at any one time, to prevent uncertain results from occurring when multiple threads attempt to change or access values included in that code at the same time.

DOS operating environment — A general term used to describe the reasonably thorough DOS emulation capabilities provided in a Windows 2000 virtual DOS machine (VDM).

dynamic link library (DLL) — A collection of virtual procedure calls, also called procedure stubs, that provide a well-defined way for applications to call on services or server processes within the Win32 environment. DLLs have been a consistent aspect of Windows since Windows 2.0.

environment subsystem — A mini-operating system running within Windows 2000, providing an interface between applications and the kernel. Windows 2000 has three environment subsystems: Win32, OS/2, and POSIX, but only Win32 is required for Windows 2000 to function.

Executive Services — A set of kernel-mode functions that controls security, system I/O, memory management, and other low-level services.

input message queue — A queue for each process, maintained by the Win32 subsystem, that contains the messages sent to the process from the user, directing its threads to do something.

kernel — The part of Windows 2000 composed of system services that interact directly with applications; it controls all application contact with the computer.

kernel mode — Systems running in kernel mode are operating within a shared memory space and with access to hardware. Windows 2000 Executive Services operates in kernel mode.

local procedure call (LPC) — A technique to permit processes to exchange data in the Windows 2000 run-time environment. LPCs define a rigorous interface to let client programs request services, and to let server programs respond to such requests.

multitasking — Sharing processor time between threads. Multitasking may be preemptive (the operating system may bump one thread if another one really needs access to the processor), or cooperative (one thread will retain control of the processor until its turn to use it is over). Windows 2000 uses preemptive multitasking except in the context of the WOW operating environment, because Windows 3.x applications expect cooperative multitasking.

multithreaded process — A process with more than one thread running at a time.

OS/2 subsystem — The Windows 2000 subsystem used for running OS/2 applications; an emulation of OS/2 version 1.x (character mode only).

parent process — The Windows 2000 environment subsystem that creates a run-time process, and imbues that child process with characteristics associated with that parent's interfaces, capabilities, and run-time requirements.

POSIX subsystem — The Windows 2000 subsystem used for running POSIX applications.

process — An environment in which the executable portion of a program runs, defining its memory usage, which processor to use, its objects, and so forth. All processes have at least one thread. When the last thread is terminated, the process terminates with it. Each user-mode process maintains its own map of the virtual memory area. One process may create another, in which case the creator is the parent process and the created process is the child process.

real mode — A DOS term that describes a mode of operation for x86 CPUs wherein they can address only 1 MB of memory, broken into 16 64-KB segments, where the lower ten such segments are available to applications (the infamous 640 KB), and the upper six segments are available to the operating system or to special application drivers—or, for Windows 2000, to a VDM.

subsystem — An operating environment that emulates another operating system (such as OS/2 or POSIX) to provide support for applications created for that environment.

synchronization object — Any of a special class of objects within the Windows 2000 environment that are used to synchronize and control access to shared objects and critical sections of code.

thread — The executable portion of a program, with a priority based on the priority of its process—user threads cannot exist external to a process. All threads in a process share that process's context.

user mode — Systems running in user mode are operating in virtual private memory areas for each process, so that each process is protected from all others. User-mode processes may not manipulate hardware, but must send requests to kernel-mode services to do this manipulation for them.

virtual device driver (VDD) — A device driver used by virtual DOS machines (VDMs) to provide an interface between the application, which expects to interact with a 16-bit device driver, and the 32-bit device drivers that Windows 2000 provides.

virtual DOS machine (VDM) — A Win32 application that emulates a DOS environment for use by DOS and Win16 applications.

Win16 operating environment — The collection of components, interfaces, and capabilities that permits Win16 applications to run within a VDM within the Win32 subsystem on Windows 2000.

Win16-on-Win32 subsystem (WOW) — The formal name for the collection of components, interfaces, and capabilities that permits the Win32 subsystem to provide native support for well-behaved 16-bit Windows applications.

Review Questions

1. Which of the following is not an environment subsystem? (Choose all correct answers.)
   a. Win32
   b. Win16
   c. OS/2
   d. none of the above
2. If the threads in a process will always run on one processor in a multiprocessor system, that process is said to have a(n) _____ for that processor.
3. Which of the following statements about process termination are true? (Choose all correct answers.)
   a. When a process's last thread is terminated, the process will terminate as well, unless it creates another thread within a certain interval.
   b. When a process terminates, all of its child processes terminate with it.
   c. A process must have at least one thread at all times.
   d. If a parent process terminates, its threads may be taken over by a child process.

4. Which of the following is not a reason to use the environment subsystem/kernel model?
   a. speed
   b. modularity
   c. subsystem protection
   d. ease of communication
5. The _____ subsystem is required for the functioning of the Windows 2000 operating system.
6. Applications and the subsystems in which they run have a _____ relationship, in that the client application asks the server subsystem to do things for it, and the subsystem complies.
7. When an application stops operating in user mode and begins operating in kernel mode, this is called a context switch. True or False?
8. Which of the following is not an attempt to speed up subsystem/user application communications?
   a. LPCs
   b. caching services provided by the subsystem
   c. batching messages
   d. calling kernel services directly
9. Which two parts of the kernel were part of the Win32 subsystem prior to Windows NT 4.0?
   a. GDI
   b. I/O Manager
   c. device drivers
   d. Windows Manager
10. User applications always operate in user mode. True or False?
11. To restart the OS/2 subsystem, you must:
    a. log off and log back on
    b. stop Os2.exe and then start a new OS/2 application
    c. restart the computer
    d. none of the above
12. The Windows 2000 Executive Services define the kernel mode of this operating system and its run-time environment. True or False?
13. POSIX applications must always run on an NTFS partition. True or False?
14. Each time you start a DOS application under Windows 2000, it runs in its own VDM. True or False?

15. Which of the following statements are true regarding LPCs, or Local Procedure Calls? (Choose all correct answers.)
    a. used to inform the CPU of I/O
    b. code optimized for speed
    c. supports specialized message passing functions
    d. employed only by the GDI portion of the OS
16. A child process can inherit the security token of its parent, or it can obtain a new security token by querying the Security Accounts Manager. True or False?
17. Windows 16-bit applications rely upon which of the following? (Choose all correct answers.)
    a. NTVDM
    b. POSIX
    c. WOW
    d. Win32
18. Under Windows 2000, what do Win16 applications all share by default? (Choose all correct answers.)
    a. working directory
    b. message queue
    c. address space
    d. NTFS file permissions
19. Which of the following are true statements about Win16? (Choose all correct answers.)
    a. Wowexec.exe functions directly within a Win32 VM.
    b. When Wowexec.exe terminates, its NTVDM host also terminates.
    c. A different kernel driver is used on RISC systems for the WOW environment.
    d. Only a single instance of Wowexec.exe can be launched.
20. Win16 applications can be launched into a separate memory space from the Run command within Windows 2000. True or False?
21. In Windows 2000, DOS applications can be launched into DOS environments with customized memory configurations. True or False?
22. All multithreaded applications running under Windows 2000 could be designed to operate on multiple processors for greater efficiency. True or False?
23. The first 16-bit application always runs in a separate memory space by default, whereas all subsequent 16-bit applications run by default in the same memory space as other applications of their kind. True or False?
24. A section of code that modifies data structures used by several threads is called a _____.
25. Neither AUTOEXEC.BAT nor CONFIG.SYS has any role in determining Windows 2000 system configuration. True or False?

HANDS-ON PROJECTS

Project 12-1

To start a Win16 application in its own address space:

1. Open Windows Explorer by selecting **Start**, **Programs**, **Accessories**, **Windows Explorer**.
2. In the left pane, select the drive letter that hosts your Windows 2000 main directory.
3. In the right pane, double-click the Windows 2000 main folder (this is WINNT by default). If necessary, click **Show Files** to see the files in this folder.
4. Scroll down in the right pane to locate Winhelp.exe.
5. Select **Winhelp.exe**.
6. Right-click over **Winhelp.exe** and select **Create Shortcut** from the resulting menu.
7. Select **Shortcut to Winhelp.exe**.
8. Right-click over **Shortcut to Winhelp.exe** and select **Properties** from the resulting menu.
9. Mark the **Run in separate memory space** check box (see Figure 12-15).

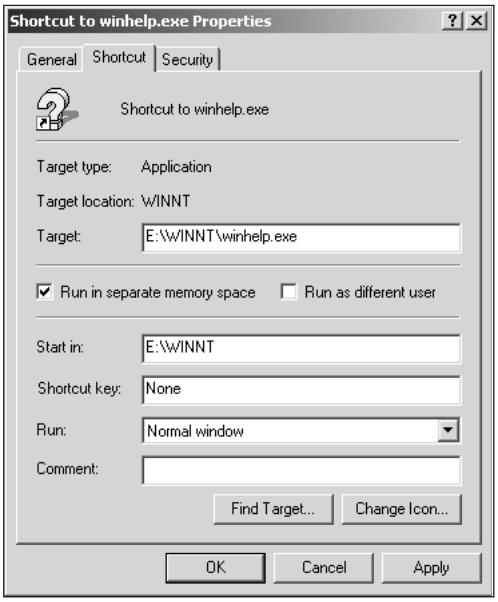

Figure 12-15 Selecting the Run in separate memory space check box

10. Click **OK**.
11. Double-click **Winhelp**.
12. Double-click **Shortcut to Winhelp.exe**.

13. Launch Task Manager by pressing **Ctrl+Shift+Esc**.
14. Select the **Processes** tab.
15. Notice that two WOW environments exist, each hosting an instance of Winhelp.exe
16. Close Task Manager by selecting **File**, **Exit Task Manager**.
17. Close both instances of Winhelp by selecting **File**, **Exit**.

Project 12-2

To explore the Properties configuration for a DOS application:

1. Open Windows Explorer by selecting **Start**, **Programs**, **Accessories**, **Windows Explorer**.
2. Locate the DOS application, **Edit.com**.
3. Right-click **Edit.com** and select **Properties** from the resulting menu.
4. View the details provided on the **General** tab (refer to Figure 12-8).
5. Select the **Program** tab and view the details provided.
6. Click the **Advanced** button and view the details provided.
7. Click **Cancel**.
8. Select the **Font** tab and view the details provided.
9. Select the **Memory** tab and view the details provided.
10. Select the **Screen** tab and view the details provided.
11. Select the **Misc** tab and view the details provided.
12. Explore other tabs, if any, on your Properties dialog box.
13. Click **Cancel** to close the Properties dialog box and discard any mistaken changes.

Project 12-3

To view the effects of various VDM-based applications on Windows 2000:

1. Restart your Windows 2000 system and log on.
2. Launch Task Manager by pressing **Ctrl+shift+esc**.
3. Select the **Processes** tab (refer to Figure 12-2).
4. Launch a DOS application, such as Edit.com, by using the **Run** command. This is performed by clicking the **Start** button then selecting **Run**. Type **c:\WINNT\System32\Edit.com** or a similar path to a DOS application. Click **OK**.
5. Notice in the list of processes via Task Manager that NTVDM has appeared, but the name of the DOS application itself has not.
6. Close the DOS application, using its own commands. For Edit.com this means selecting **File**, **Exit**.
7. Once the application terminates, notice that NTVDM no longer appears in the process list.

8. Launch a Windows 16 bit application, such as Winhelp.exe, by using the Run command. This is performed by clicking the **Start** button then selecting **Run**. Type **c:\winnt\Winhelp.exe** or a similar path to a Win16 application. Click **OK**.
9. Notice in the list of processes via Task Manager that NTVDM appears along with Wowexec.exe and Winhelp.exe as subitems.
10. Close the Win16 application using its own commands. For Winhelp.exe, this means selecting **File**, **Exit**.
11. Notice in the list of processes that NTVDM and WOWEXEC remain.
12. Terminate the WOWEXEC process by selecting it and clicking **End Process**.
13. Click **Yes** to confirm termination.
14. Notice that both NTVDM and WOWEXEC are no longer listed in the processes.
15. Close Task Manager by selecting **File**, **Exit Task Manager**.

Project 12-4

To view the AUTOEXEC.NT and CONFIG.NT files of Windows 2000:

1. Launch Notepad by selecting **Start**, **Programs**, **Accessories**, **Notepad**.
2. Select **File**, **Open**.
3. Change directories to **\WINNT\System32**.
4. Change the **Files of type** to **All Files**.
5. Locate and select **AUTOEXEC.NT**.
6. Click **Open**.
7. View the contents of this file (refer to Figure 12-5).
8. Select **File**, **Open**.
9. Change the **Files of type** to **All Files**.
10. Locate and select **CONFIG.NT**.
11. Click **Open** and view the contents of this file (refer to Figure 12-6).
12. Close Notepad by selecting **File**, **Exit**.

Project 12-5

To view the number of threads used by processes under Windows 2000:

1. Launch Task Manager by clicking **ctrl+shift+Esc**.
2. Select the **Processes** tab (refer to Figure 12-2).
3. Select **View**, **Select Columns**.
4. Mark the **Thread Count** check box (see Figure 12-16).

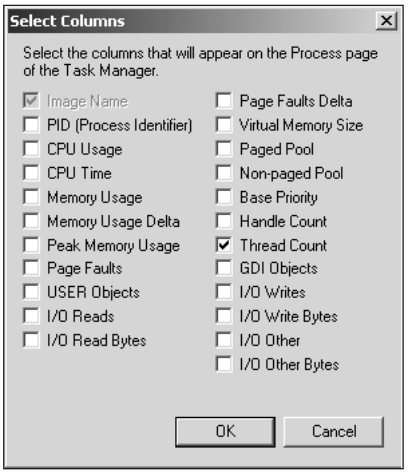

Figure 12-16 Selecting the Thread Count check box

5. Click **OK**.
6. Maximize the Task Manager window.
7. Notice the number of threads for each of the currently active processes.
8. Close Task Manager by selecting **File**, **Exit Task Manager**.

CASE PROJECTS

1. To avoid the need to reimplement old code for your user community, which is in the process of upgrading from Windows 95 to Windows 2000 Professional, you decide to allow your users to run your company's homegrown application, Teller.exe, which is a well-behaved 16-bit Windows application, on their machines. Because this program sometimes hangs for as much as 2 or 3 minutes while computing end-of-day balances, it may cause problems for other 16-bit Windows applications that your users might need to run. What can you do to insulate these other applications from Teller.exe? How might you launch this program to accomplish this goal?
2. At XYZ Corp., the company has decided to switch from its OS/2 machines to Windows 2000 Professional. Your manager informs you that this will be a snap because Windows 2000 includes an OS/2 subsystem that supports the company's homegrown graphical OS/2 applications. What must you tell your manager about his assumptions about Windows 2000 support for OS/2? Why is this a problem?
3. Given a list of DOS and 16-bit Windows applications that you may wish to use on a Windows 2000 machine, what is the proper method to ensure that each of them will (or won't) work with this operating system? What happens if any of these applications is ill behaved?